Lab 5

David Kirby

Due: 16 May 2020

Deliverables

2. (a) In our pipeline, can the data hazard below be resolved with forwarding alone? Why or why not?

```
lw $t0, 0($t1)
sw $t3, 0($t0)
```

The data hazard can be resolved with forwarding alone. This is because there is a forwarding MUX implemented in the execute stage that allows us to resolve the data hazard.

(b) Will the pipeline control unit slip the pipeline in the case of the instruction sequence above? Why or why not?
Yes, the pipeline control unit will slip the pipeline in the case of the instruction sequence above as it is designed to stall the fetch and decode stages, allowing the load to proceed down the pipeline.

(c) Which components in Figure 1 does the test bench file, MIPSpipeline_tb.vhd, provide?
Instruction memory, jump targets, and loops.

(d) Determine the CPI of the test code running on your pipelined processor. Compare this CPI to that of the single-cycle processor.
I unfortunately wasn't able to complete the processor to compare.

Source Code

MIPSpipeline.vhd

```
1  ----------------------------------------------------------------------------------
2  -- This VHDL source file describes the basic 5-stage pipelined processor
3  -- discussed in David Patterson and John Hennessy's Computer Organization and
4  -- Design textbook. Instruction processing goes through the following stages:
5  --IF --- Instruction fetch: Here instructions are fetch, one per clock cycle,
6  -- from the memory pointed to by the Program Counter (PC).
7  --ID --- Instruction decode: The instruction fetched in the prevous stage is
8  -- decoded into a number of control signals that flow down the pipe.
9  -- Also, the register file is read in this stage. Data and control
10 -- dependencies are detected and handled in this stage, either with
11 -- forwarding or interlocking.
12 --EX --- Execute: The Arithmetic-Logic Unit (ALU) is in this stage. Also,
13 -- branches are resolved in the execute stage.
14 --MEM --- Memory: In this stage, memory is accessed. For a load, memory is
15 -- read, while for a store, memory is written.
16 --WB --- Write Back: This is the final stage of the pipeline. Here the
17 -- register file is written.
18 --
```

```
--The logic described in this src file attempts to follow the natural flow of
--the pipeline. PC logic and other instruction fetch related HW is described
--at the beginning, while Write-back logic is at the end.
--
--The signals in the pipeline attempt to adhere to the following format:
--prefix_signal_name_suffix
--A prefix identifies the location within the pipeline where a given signal
--comes from. If two stages are specified for the prefix, then the signal
--comes from the pipeline register between those two stages. For example,
--ID_EX_load is the load signal from the ID/EX pipeline register. If only one
--stage is specified in the prefix, the signal originates from the given
--stage. For example, ID_load originates from the decode unit within the ID
--stage. If no prefix exists, that signal is assumed to be contained within
--only one pipeline stage.
--A suffix indicates the destination of a signal and is added to a signal name
--only when necessary. Most signals are used in multiple stages and therefore
--have no specific destination. However, some signals have the same signal
--name but are expected to used in different stages. For example,
--DataMemForwardCtrl_EX and DataMemForwardCtrl_MEM are different signals,
--serving similar functions but destine for different stages.
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --needed to describe
use ieee.std_logic_arith.all; --arithmetic
----------------------------------------------------------------------------------

entity MIPSpipeline is

    port (
        Clk, Rst_L : in std_logic;
        PC : out std_logic_vector(31 downto 0);
        Instruction : in std_logic_vector(31 downto 0);
        DataMemAddr : out std_logic_vector(31 downto 0);
        DataMemRdEn, DataMemWrEn : out std_logic;
        DataMemRdData : in std_logic_vector(31 downto 0);
        DataMemWrData : out std_logic_vector(31 downto 0)
    );
end MIPSpipeline;

----------------------------------------------------------------------------------

architecture MIPSpipeline of MIPSpipeline is

    -- Component declarations go here...

    component RegisterFile is
        port (
            RdRegA : in std_logic_vector(4 downto 0);
            RdRegB : in std_logic_vector(4 downto 0);
            WrReg : in std_logic_vector(4 downto 0);
            Clk : in std_logic;
            RegWrEn : in std_logic;
```

```
73              WrData : in std_logic_vector(31 downto 0);
74              RdDataA : out std_logic_vector(31 downto 0);
75              RdDataB : out std_logic_vector(31 downto 0)
76          );
77      end component;
78
79      component ALU is
80          port (
81              AluCtrl : in std_logic_vector(3 downto 0);
82              AluInA, AluInB : in std_logic_vector(31 downto 0);
83              AluResult : out std_logic_vector(31 downto 0)--;
84              Equals : out std_logic
85          );
86      end component;
87
88      component AluDecode is
89
90          port (
91              AluOp : in std_logic_vector(3 downto 0);
92              Funct : in std_logic_vector(5 downto 0);
93              AluCtrl : out std_logic_vector(3 downto 0)
94          );
95      end component;
96
97      component InstrDecode is
98
99          port (
100             --6-bit op field of instruction
101             Opcode : in std_logic_vector(5 downto 0);
102             --6-bit funct field of instruction
103             Funct : in std_logic_vector(5 downto 0);
104             -- asserted when shamt field is input to ALU
105             UseShamt : out std_logic;
106             -- asserted when the immediate field is input to ALU
107             UseImmed : out std_logic;
108             -- asserted when immed needs to be sign extended
109             SignExtend : out std_logic;
110             -- asserted when instruction is a jump
111             Jump : out std_logic;
112             -- asserted when instruction is a branch
113             Branch : out std_logic;
114             -- Determines ALU operation, see ALU decode unit
115             AluOp : out std_logic_vector(3 downto 0);
116             -- selects between rt and rd for the register destination
117             -- rt = 0, rd = 1
118             RegDst : out std_logic;
119             -- asserted when reading from memory (loads!)
120             MemRdEn : out std_logic;
121             -- asserted when writing to memory (stores!)
122             MemWrEn : out std_logic;
123             -- selects the source for writing into register file
124             RegSrc : out std_logic;
125             -- asserted when writing into register file
126             RegWrEn : out std_logic
```

```
127             -- that's all folks!
128         );
129     end component;
130 
131     component ForwardingUnit is
132 
133         port (
134             UseShamt, UseImmed : in std_logic;
135             EX_RegWrEn, MEM_RegWrEn : in std_logic;
136             ID_Rs, ID_Rt : in std_logic_vector(4 downto 0);
137             EX_WrReg, MEM_WrReg : in std_logic_vector(4 downto 0);
138             AluSrcA, AluSrcB : out std_logic_vector(1 downto 0);
139             DataMemForwardCtrl_EX : out std_logic;
140             DataMemForwardCtrl_MEM : out std_logic
141         );
142     end component;
143 
144     component PipelineCtrl is
145 
146         port (
147             EX_Branch : in std_logic;
148             EX_Equals : in std_logic;
149             ID_Jump : in std_logic;
150             ID_Rs, ID_Rt : std_logic_vector(4 downto 0);
151             UseShamt, UseImmed : in std_logic;
152             EX_MemRdEn : in std_logic;--to detect a load
153             EX_WrReg : in std_logic_vector(4 downto 0);
154             PCwrite : out std_logic;
155             addrSel : out std_logic_vector(1 downto 0);
156             Flush_IF_ID, WrEn_IF_ID : out std_logic;
157             Flush_ID_EX : out std_logic
158         );
159     end component;
160 
161     -- Declare signals and variables here...
162     signal Clk, Rst_L : std_logic;
163 
164     -- Typically, the code is easier to read if signals are
165     -- declared in the order they appear below...
166 
167     -- Instruction fetch signals
168     signal IF_ID_Instruction : std_logic_vector(31 downto 0);
169     -- Pipeline control signals
170     signal Flush_IF_ID, WrEn_IF_ID : std_logic;
171 
172     -- Instruction decode signals
173     signal ID_Rs, ID_Rt : std_logic_vector(4 downto 0);
174 
175     -- Execute stage signals
176     signal Shamt : std_logic_vector(31 downto 0);
177     signal AluInA : std_logic_vector(31 downto 0);
178     signal AluInB : std_logic_vector(31 downto 0);
179     signal EX_AluResult : std_logic_vector(31 downto 0);
180     signal EX_Equals : std_logic;
```

```
181     signal EX_RegDst : std_logic;
182     signal EX_DataMemWrData : std_logic_vector(31 downto 0);
183     signal EX_WrReg : std_logic_vector(4 downto 0);
184 
185     --EX/MEM pipeline registers
186     signal EX_MEM_MemRdEn, EX_MEM_MemWrEn : std_logic;
187     signal EX_MEM_RegSrc : std_logic;
188     signal EX_MEM_RegWrEn : std_logic;
189     signal EX_MEM_AluResult : std_logic_vector(31 downto 0);
190     signal EX_MEM_DataMemWrData : std_logic_vector(31 downto 0);
191     signal EX_MEM_WrReg : std_logic_vector(4 downto 0);
192     signal EX_MEM_DataMemForwardCtrl_MEM : std_logic;
193 
194     --MEM/WB pipeline registers
195     signal MEM_WB_RegSrc : std_logic;
196     signal MEM_WB_RegWrEn : std_logic;
197     signal MEM_WB_DataMemRdData : std_logic_vector(31 downto 0);
198     signal MEM_WB_AluResult : std_logic_vector(31 downto 0);
199     signal MEM_WB_WrReg : std_logic_vector(4 downto 0);
200 
201     -- Write back stage signals
202     signal WB_RegWrData : std_logic_vector(31 downto 0);
203 
204     ----------------------------------------------------------------------------------
205 begin
206 
207     ---------------------------------------------------
208     -- Instruction Fetch logic
209 
210     -- insert PC logic here, marking the beginning of the fetch stage
211 
212     -- IF_ID pipeline registers mark the end of the fetch stage
213     process (Clk)
214     begin --rising edge triggered logic
215         if (Clk'event and Clk = '1') then
216             if (Flush_IF_ID = '1') then --active high, synchronous reset
217                 IF_ID_Instruction <= (others => '0');
218                 --Add more registers here
219             elsif (WrEn_IF_ID = '1') then
220                 IF_ID_Instruction <= Instruction;
221                 --Add more registers here
222             end if;
223         end if;
224     end process;
225     ---------------------------------------------------
226     -- Instruction Decode logic
227 
228     --break up instruction into parts for readability
229     ID_Rs <= IF_ID_Instruction(25 downto 21);
230     ID_Rt <= IF_ID_Instruction(20 downto 16);
231 
232     --instantiate the register file
233     --instance_name: component_name
234     -- use nominal port map as opposed to positional!
```

```
235     Registers : RegisterFile
236     port map(
237         RdRegA => ID_Rs,
238         RdRegB => ID_Rt,
239         WrReg => MEM_WB_WrReg,
240         Clk => Clk,
241         WrData => WB_RegWrData,
242         --okay you get the point...
243     );
244 
245     ----------------------------------------------------
246     -- Execute stage logic
247 
248     --zero extend the shift amount field
249     Shamt(4 downto 0) <= ID_EX_Instruction(10 downto 6);
250     Shamt(31 downto 5) <= (others => '0');
251 
252     -- ALU source mux for OpA
253     --The WITH/SELECT/WHEN construct is great for muxes
254     with ID_EX_AluSrcA select
255         -- the zero extended shamt field
256         AluInA <= Shamt when "00",
257         -- the bypass path from the write back stage
258         WB_RegWrData when "01",
259         -- the bypass path from the memory stage
260         EX_MEM_AluResult when "10",
261         -- the register file output
262         ID_EX_RdRegA when others;--avoids latches!
263 
264     -- ALU should be instantiated somewhere here...
265 
266     -- EX/MEM pipeline registers here...
267 
268     ----------------------------------------------------
269     -- Memory stage logic
270     -- DataMemWrData mux here...
271     -- MEM/WB pipeline registers here...
272 
273 end MIPSpipeline;
```